Sparse Table

Bruce

What is a sparse table?

- Sparse Table is a data structure that answers **static Range Minimum Query (RMQ).**

- It is recognized for its relatively **fast query** and **short implementation** compared to other data structures.

- Applications:
  - Range Minimum Query (Range min/max query)
  - Lowest Common Ancestor Query (LCA)

Range Minimum Queries

The RMQ Problem

- The ***Range Minimum Query problem*** (***RMQ*** for short) is the following:

  Given an array A and two indices $i \leq j$, what is the smallest element out of $A[i], A[i+1], \ldots, A[j-1], A[j]$?

31	41	59	26	53	58	97	93

The RMQ Problem

- The ***Range Minimum Query problem*** (***RMQ*** for short) is the following:

  Given an array A and two indices $i \leq j$, what is the smallest element out of $A[i], A[i+1], \ldots, A[j-1], A[j]$?

31	41	59	26	53	58	97	93

The RMQ Problem

- The ***Range Minimum Query problem*** (***RMQ*** for short) is the following:

Given an array A and two indices $i \leq j$, what is the smallest element out of A[i], A[i + 1], …, A[j – 1], A[j]?

31	41	59	26	53	58	97	93

Methods

- Naïve way: no preprocessing, O(n) query
- Cache everything: $O(n^2)$ preprocessing, O(1) query
- Segment Tree: O(nlogn) preprocessing, O(logn) query
- Sparse Table: O(nlogn) preprocessing, O(1) query

Cache every pairs

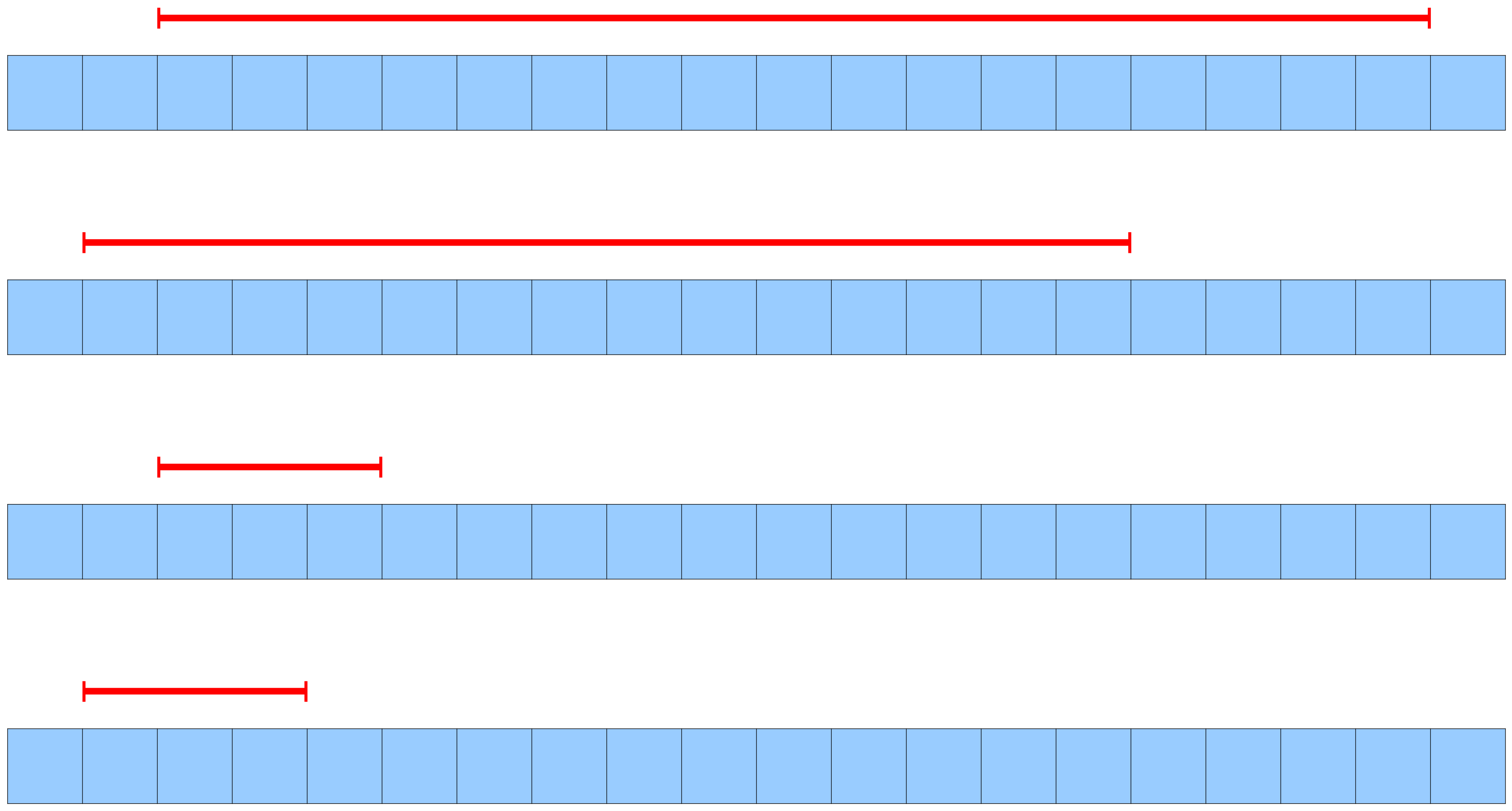
31	41	59	26	53	58	97	93
0	1	2	3	4	5	6	7

	0	1	2	3	4	5	6	7
0	31	31	31	26	26	26	26	26
1		41	41	26	26	26	26	26
2			59	26	26	26	26	26
3				26	26	26	26	26
4					53	53	53	53
5						58	58	58
6							97	93
7								93

The Intuition

- If we precompute the answers over too many ranges, the preprocessing time will be too large.
- If we precompute the answers over too few ranges, the query time won't be $O(1)$.
- ***Goal:*** Precompute RMQ over a set of ranges such that there are fewer than $o(n^2)$ total ranges, but there are enough ranges to support $O(1)$ query.

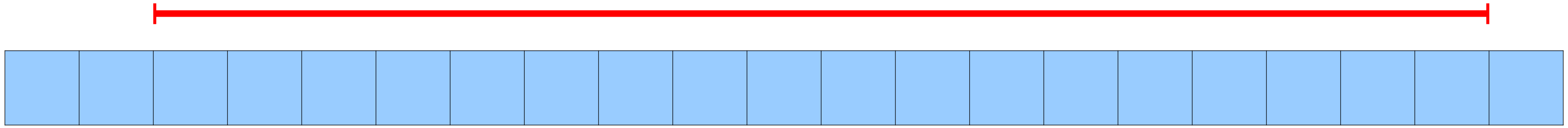
Some Observations



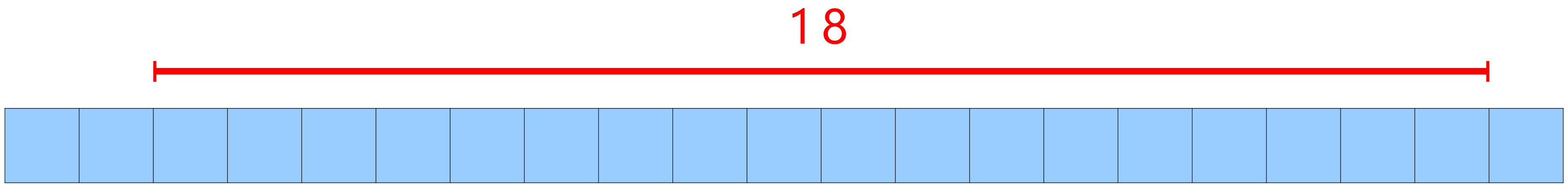
The Approach

- For each index i, compute RMQ for ranges starting at i of size 1, 2, 4, 8, 16, …, 2^k as long as they fit in the array.
  - Gives both large and small ranges starting at any point in the array.
  - Only O(log n) ranges computed for each array element.
  - Total number of ranges: O(n log n).
- ***Claim:*** Any range in the array can be formed as the union of two of these ranges.

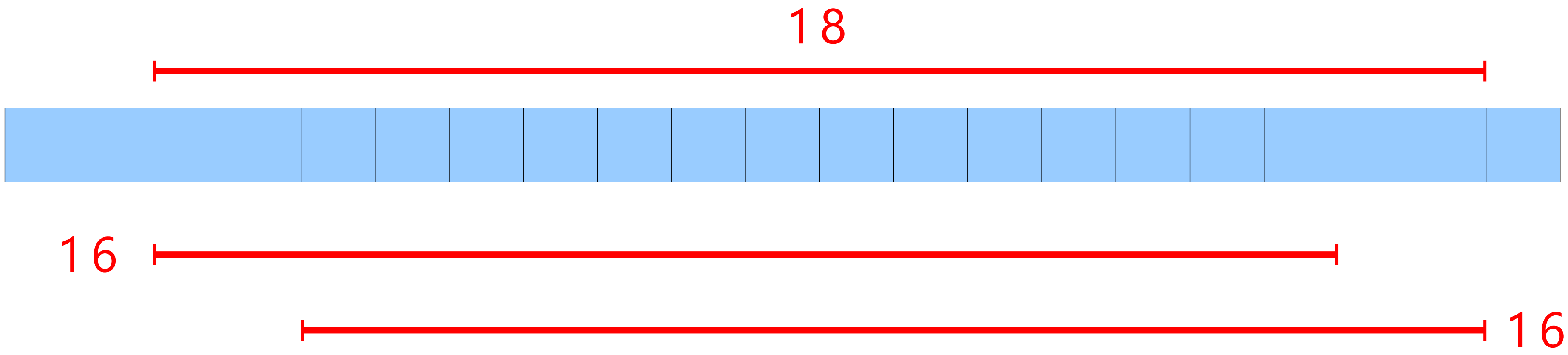
Creating Ranges



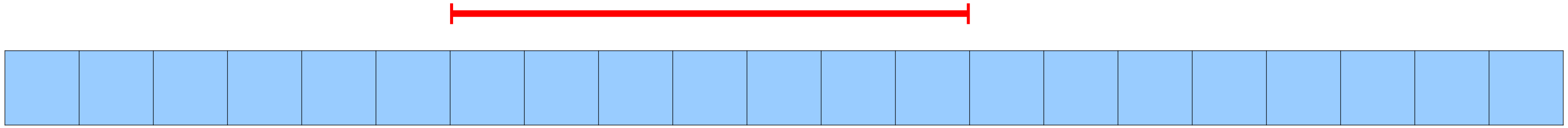
Creating Ranges



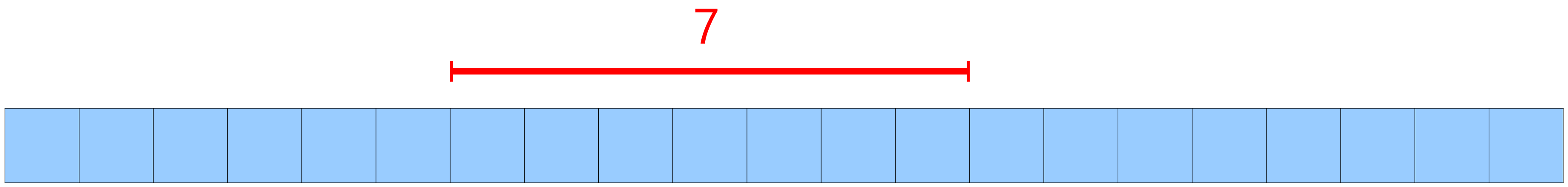
Creating Ranges



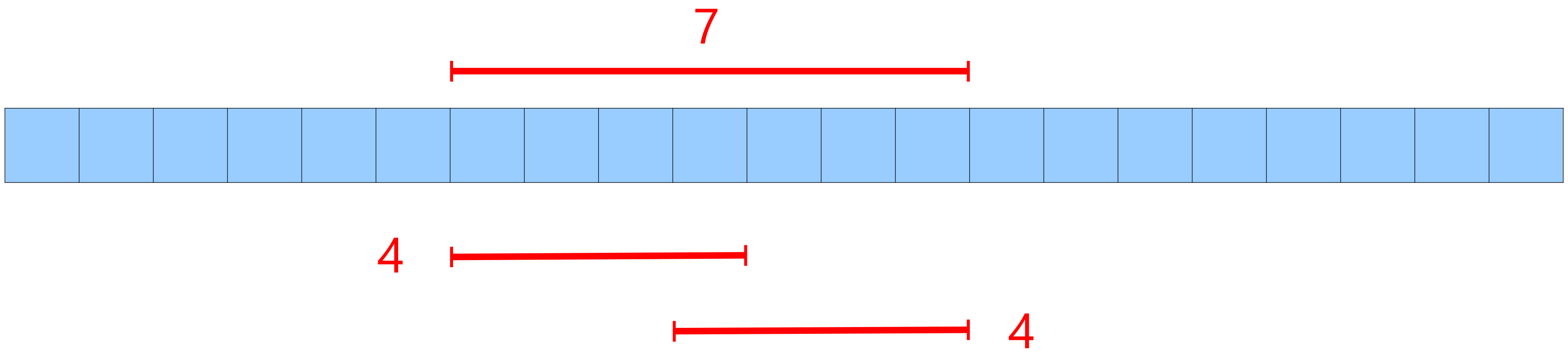
Creating Ranges



Creating Ranges



Creating Ranges



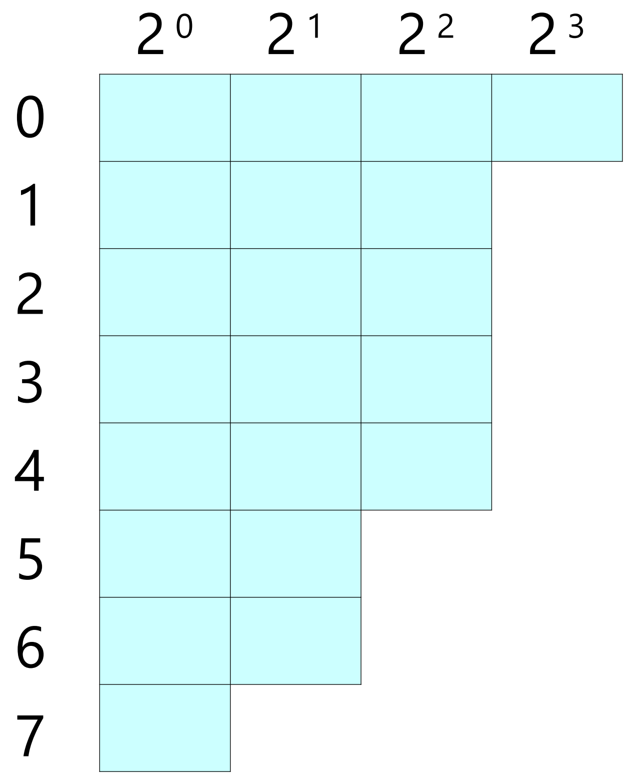
Doing a Query

- To answer $RMQ_A(i, j)$:
  - Find the largest k such that $2^k \leq j - i + 1$.
    - With the right preprocessing, this can be done in time O(1); you'll figure out how in the problem set!
  - The range $[i, j]$ can be formed as the overlap of the ranges $[i, i + 2^k - 1]$ and $[j - 2^k + 1, j]$.
  - Each range can be looked up in time O(1).
  - Total time: O(1).

Precomputing the Ranges

- There are O(*n* log *n*) ranges to precompute.
- Using dynamic programming, we can compute all of them in time O(*n* log *n*).

31	41	59	26	53	58	97	93
0	1	2	3	4	5	6	7



Precomputing the Ranges

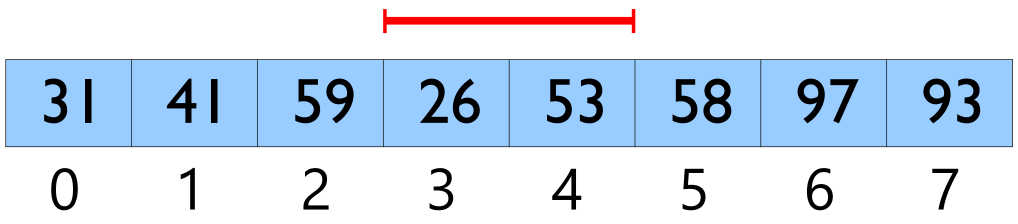
- There are O(n log n) ranges to precompute.
- Using dynamic programming, we can compute all of them in time O(n log n).

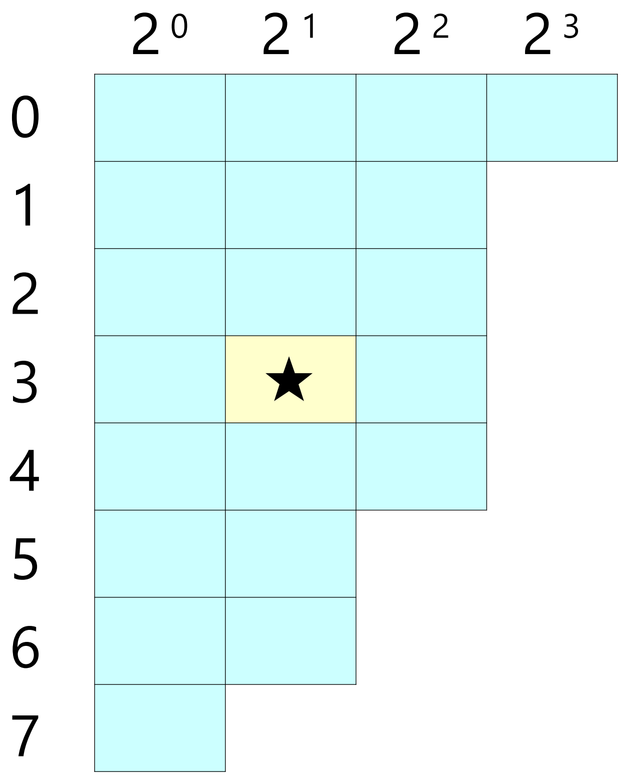
	2^0	2^1	2^2	2^3
0				
1				
2				
3		★		
4				
5				
6				
7				

31	41	59	26	53	58	97	93
0	1	2	3	4	5	6	7

Precomputing the Ranges

- There are $O(n \log n)$ ranges to precompute.
- Using dynamic programming, we can compute all of them in time $O(n \log n)$.

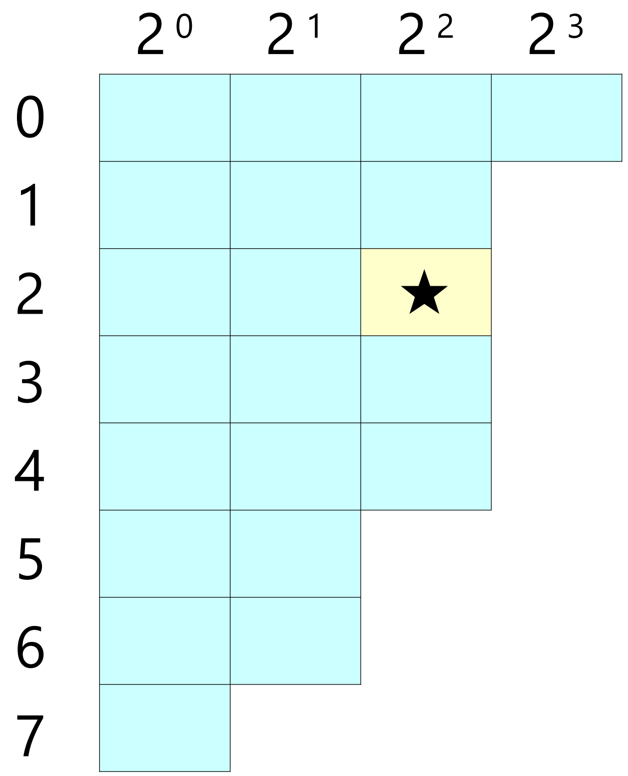




Precomputing the Ranges

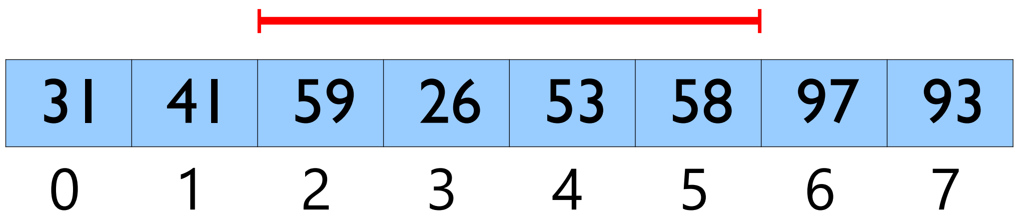
- There are $O(n \log n)$ ranges to precompute.
- Using dynamic programming, we can compute all of them in time $O(n \log n)$.

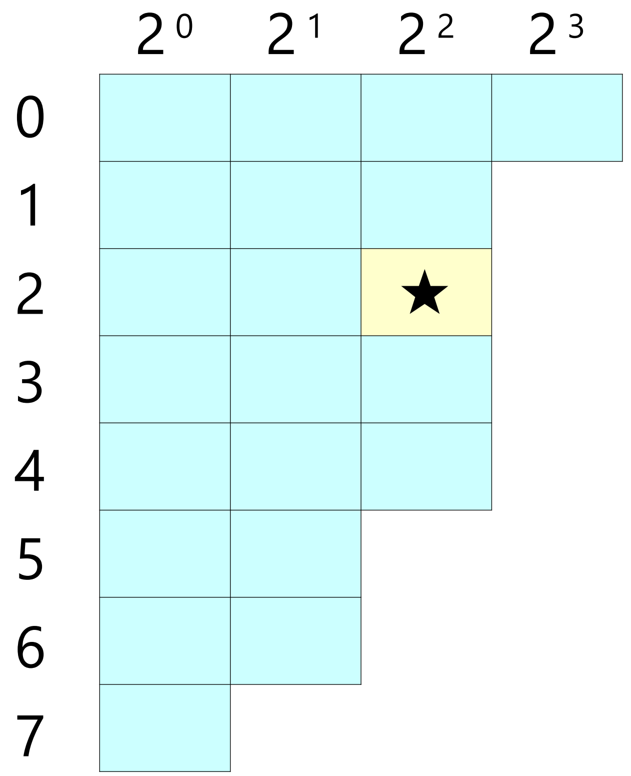
31	41	59	26	53	58	97	93
0	1	2	3	4	5	6	7



Precomputing the Ranges

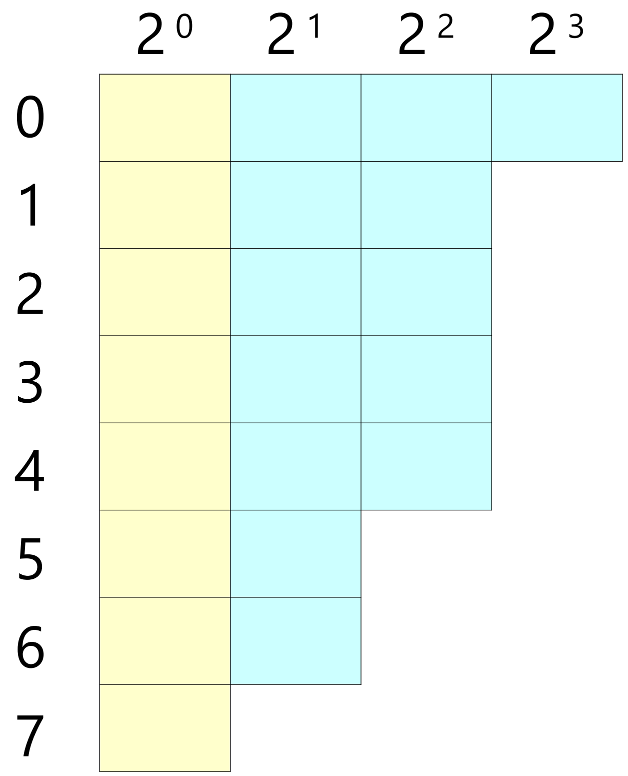
- There are $O(n \log n)$ ranges to precompute.
- Using dynamic programming, we can compute all of them in time $O(n \log n)$.





Precomputing the Ranges

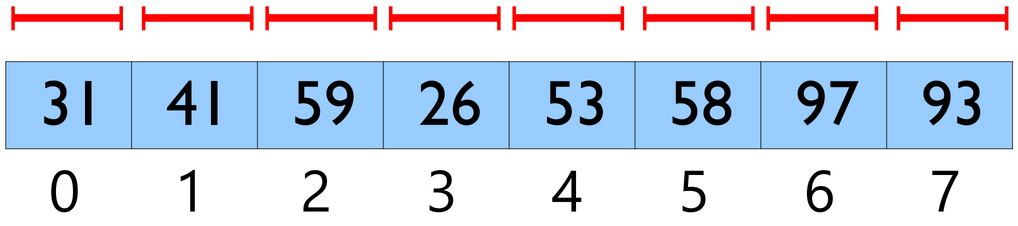
- There are $O(n \log n)$ ranges to precompute.
- Using dynamic programming, we can compute all of them in time $O(n \log n)$.

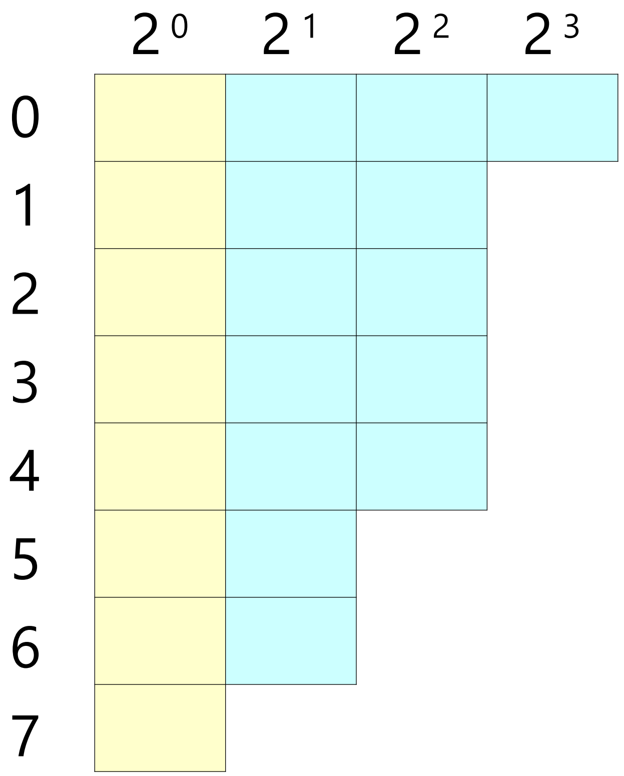


31	41	59	26	53	58	97	93
0	1	2	3	4	5	6	7

Precomputing the Ranges

- There are $O(n \log n)$ ranges to precompute.
- Using dynamic programming, we can compute all of them in time $O(n \log n)$.





Precomputing the Ranges

- There are O(n log n) ranges to precompute.
- Using dynamic programming, we can compute all of them in time O(n log n).

0	1	2	3	4	5	6	7
31	41	59	26	53	58	97	93

	2^0	2^1	2^2	2^3
0	31			
1	41			
2	59			
3	26			
4	53			
5	58			
6	97			
7	93			

Precomputing the Ranges

- There are O(n log n) ranges to precompute.
- Using dynamic programming, we can compute all of them in time O(n log n).

	2^0	2^1	2^2	2^3
0	31			
1	41			
2	59			
3	26			
4	53			
5	58			
6	97			
7	93			

31	41	59	26	53	58	97	93
0	1	2	3	4	5	6	7

Precomputing the Ranges

- There are O(n log n) ranges to precompute.
- Using dynamic programming, we can compute all of them in time O(n log n).

31	41	59	26	53	58	97	93
0	1	2	3	4	5	6	7

	2^0	2^1	2^2	2^3
0	31	★		
1	41			
2	59			
3	26			
4	53			
5	58			
6	97			
7	93			

Precomputing the Ranges

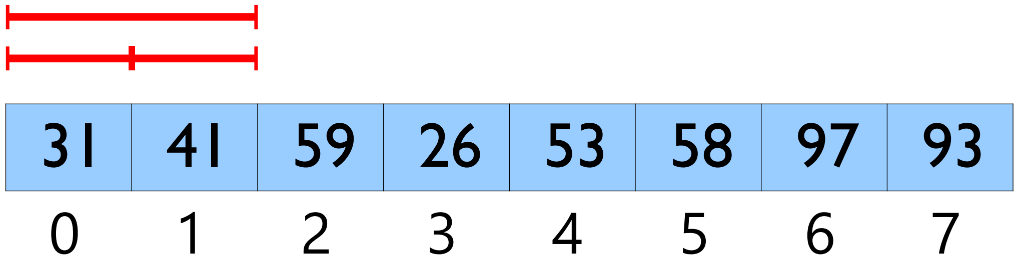
- There are $O(n \log n)$ ranges to precompute.
- Using dynamic programming, we can compute all of them in time $O(n \log n)$.

31	41	59	26	53	58	97	93
0	1	2	3	4	5	6	7

	2^0	2^1	2^2	2^3
0	31	★		
1	41			
2	59			
3	26			
4	53			
5	58			
6	97			
7	93			

Precomputing the Ranges

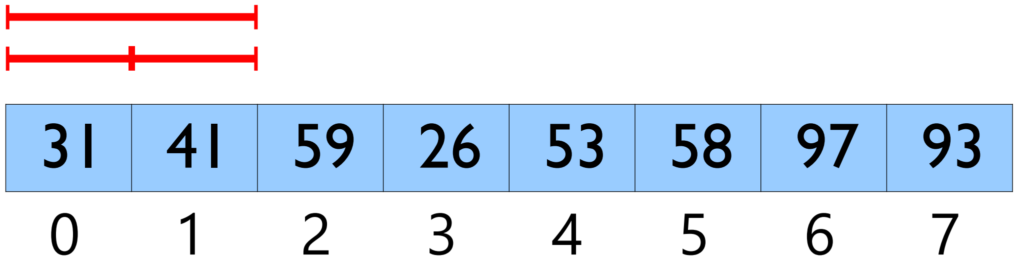
- There are $O(n \log n)$ ranges to precompute.
- Using dynamic programming, we can compute all of them in time $O(n \log n)$.



	2^0	2^1	2^2	2^3
0	31	★		
1	41			
2	59			
3	26			
4	53			
5	58			
6	97			
7	93			

Precomputing the Ranges

- There are O(n log n) ranges to precompute.
- Using dynamic programming, we can compute all of them in time O(n log n).



	2^0	2^1	2^2	2^3
0	31	★		
1	41			
2	59			
3	26			
4	53			
5	58			
6	97			
7	93			

Precomputing the Ranges

- There are O(n log n) ranges to precompute.
- Using dynamic programming, we can compute all of them in time O(n log n).

31	41	59	26	53	58	97	93
0	1	2	3	4	5	6	7

	2^0	2^1	2^2	2^3
0	31	31		
1	41			
2	59			
3	26			
4	53			
5	58			
6	97			
7	93			

Precomputing the Ranges

- There are $O(n \log n)$ ranges to precompute.
- Using dynamic programming, we can compute all of them in time $O(n \log n)$.

	2^0	2^1	2^2	2^3
0	31	31		
1	41			
2	59			
3	26			
4	53			
5	58			
6	97			
7	93			

31	41	59	26	53	58	97	93
0	1	2	3	4	5	6	7

Precomputing the Ranges

- There are $O(n \log n)$ ranges to precompute.
- Using dynamic programming, we can compute all of them in time $O(n \log n)$.

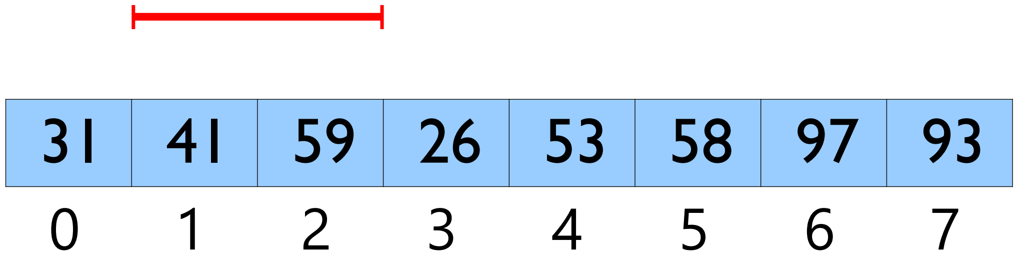
	2^0	2^1	2^2	2^3
0	31	31		
1	41	★		
2	59			
3	26			
4	53			
5	58			
6	97			
7	93			

31	41	59	26	53	58	97	93
0	1	2	3	4	5	6	7

Precomputing the Ranges

- There are $O(n \log n)$ ranges to precompute.
- Using dynamic programming, we can compute all of them in time $O(n \log n)$.

	2^0	2^1	2^2	2^3
0	31	31		
1	41	★		
2	59			
3	26			
4	53			
5	58			
6	97			
7	93			



Precomputing the Ranges

- There are O(n log n) ranges to precompute.
- Using dynamic programming, we can compute all of them in time O(n log n).

31	41	59	26	53	58	97	93
0	1	2	3	4	5	6	7

	2^0	2^1	2^2	2^3
0	31	31		
1	41	★		
2	59			
3	26			
4	53			
5	58			
6	97			
7	93			

Precomputing the Ranges

- There are $O(n \log n)$ ranges to precompute.
- Using dynamic programming, we can compute all of them in time $O(n \log n)$.

31	41	59	26	53	58	97	93
0	1	2	3	4	5	6	7

	2^0	2^1	2^2	2^3
0	31	31		
1	41	★		
2	59			
3	26			
4	53			
5	58			
6	97			
7	93			

Precomputing the Ranges

- There are O(n log n) ranges to precompute.
- Using dynamic programming, we can compute all of them in time O(n log n).

31	41	59	26	53	58	97	93
0	1	2	3	4	5	6	7

	2^0	2^1	2^2	2^3
0	31	31		
1	41	41		
2	59			
3	26			
4	53			
5	58			
6	97			
7	93			

Precomputing the Ranges

- There are $O(n \log n)$ ranges to precompute.
- Using dynamic programming, we can compute all of them in time $O(n \log n)$.

	2^0	2^1	2^2	2^3
0	31	31		
1	41	41		
2	59			
3	26			
4	53			
5	58			
6	97			
7	93			

31	41	59	26	53	58	97	93
0	1	2	3	4	5	6	7

Precomputing the Ranges

- There are O(n log n) ranges to precompute.
- Using dynamic programming, we can compute all of them in time O(n log n).

	2^0	2^1	2^2	2^3
0	31	31		
1	41	41		
2	59	26		
3	26	26		
4	53	53		
5	58	58		
6	97	93		
7	93			

31	41	59	26	53	58	97	93
0	1	2	3	4	5	6	7

Precomputing the Ranges

- There are O(n log n) ranges to precompute.
- Using dynamic programming, we can compute all of them in time O(n log n).

31	41	59	26	53	58	97	93
0	1	2	3	4	5	6	7

	2^0	2^1	2^2	2^3
0	31	31	★	
1	41	41		
2	59	26		
3	26	26		
4	53	53		
5	58	58		
6	97	93		
7	93			

Precomputing the Ranges

- There are $O(n \log n)$ ranges to precompute.
- Using dynamic programming, we can compute all of them in time $O(n \log n)$.

31	41	59	26	53	58	97	93
0	1	2	3	4	5	6	7

	2^0	2^1	2^2	2^3
0	31	31	★	
1	41	41		
2	59	26		
3	26	26		
4	53	53		
5	58	58		
6	97	93		
7	93			

Precomputing the Ranges

- There are O(n log n) ranges to precompute.
- Using dynamic programming, we can compute all of them in time O(n log n).

	2^0	2^1	2^2	2^3
0	31	31	★	
1	41	41		
2	59	26		
3	26	26		
4	53	53		
5	58	58		
6	97	93		
7	93			

31	41	59	26	53	58	97	93
0	1	2	3	4	5	6	7

Precomputing the Ranges

- There are O(n log n) ranges to precompute.
- Using dynamic programming, we can compute all of them in time O(n log n).

31	41	59	26	53	58	97	93
0	1	2	3	4	5	6	7

	2^0	2^1	2^2	2^3
0	31	31	★	
1	41	41		
2	59	26		
3	26	26		
4	53	53		
5	58	58		
6	97	93		
7	93			

Precomputing the Ranges

- There are $O(n \log n)$ ranges to precompute.
- Using dynamic programming, we can compute all of them in time $O(n \log n)$.

31	41	59	26	53	58	97	93
0	1	2	3	4	5	6	7

	2^0	2^1	2^2	2^3
0	31	31	26	
1	41	41		
2	59	26		
3	26	26		
4	53	53		
5	58	58		
6	97	93		
7	93			

Precomputing the Ranges

- There are O(n log n) ranges to precompute.
- Using dynamic programming, we can compute all of them in time O(n log n).

	2^0	2^1	2^2	2^3
0	31	31	26	
1	41	41		
2	59	26		
3	26	26		
4	53	53		
5	58	58		
6	97	93		
7	93			

31	41	59	26	53	58	97	93
0	1	2	3	4	5	6	7

Precomputing the Ranges

- There are $O(n \log n)$ ranges to precompute.
- Using dynamic programming, we can compute all of them in time $O(n \log n)$.

31	41	59	26	53	58	97	93
0	1	2	3	4	5	6	7

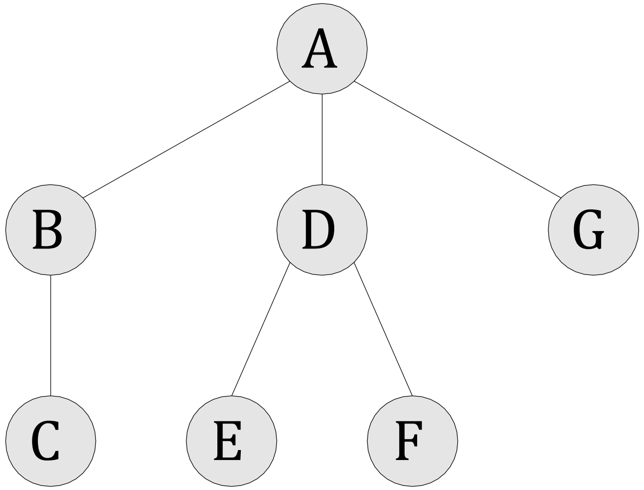
	2^0	2^1	2^2	2^3
0	31	31	26	26
1	41	41	26	
2	59	26	26	
3	26	26	26	
4	53	53	53	
5	58	58		
6	97	93		
7	93			

Sparse Tables

- This data structure is called a ***sparse table***.
- It gives an $O(n \log n)$ preprocessing, and $O(1)$ query solution to RMQ.
- This is asymptotically better than precomputing all possible ranges!

Lowest Common Ancestor

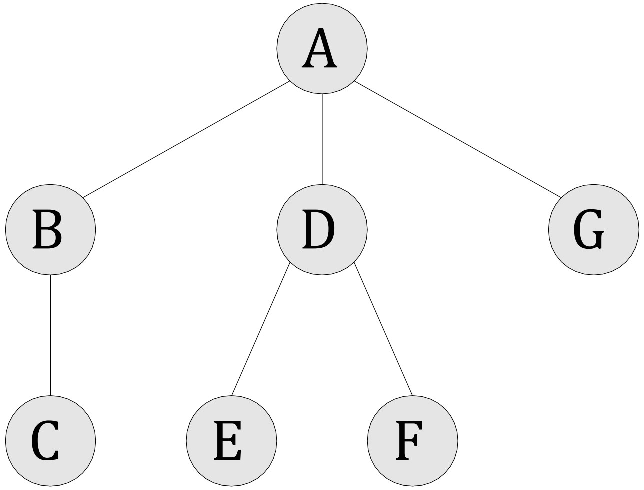
Lowest Common Ancestors



A	B	C	C	B	A	D	E	E	D	F	F	D	A	G	G	A

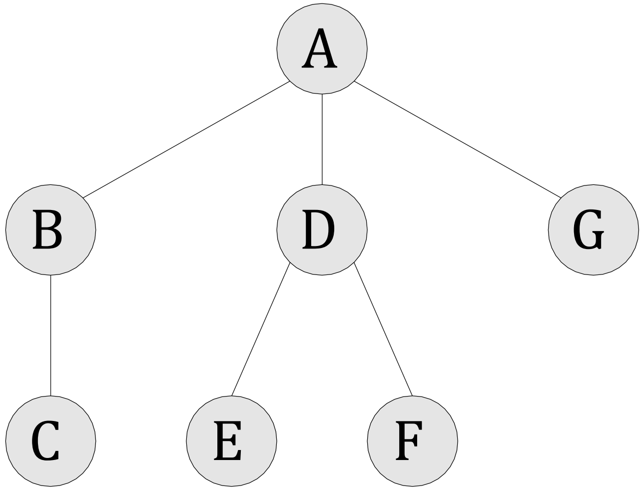
This is called an ***Euler tour*** of the tree. Euler tours have all sorts of nice properties. Depending on what topics we explore, we might see some more of them later in the quarter.

Lowest Common Ancestors



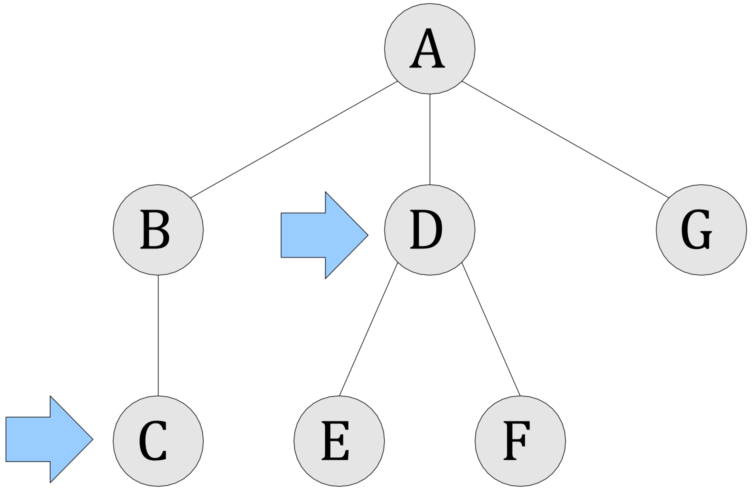
A	B	C	C	B	A	D	E	E	D	F	F	D	A	G	G	A
0	1	2	2	1	0	1	2	2	1	2	2	1	0	1	1	0

Lowest Common Ancestors



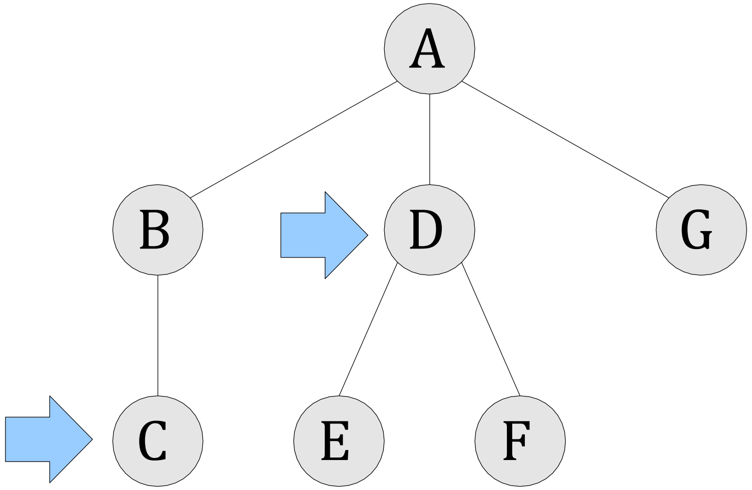
A	B	C	C	B	A	D	E	E	D	F	F	D	A	G	G	A
0	1	2	2	1	0	1	2	2	1	2	2	1	0	1	1	0

Lowest Common Ancestors



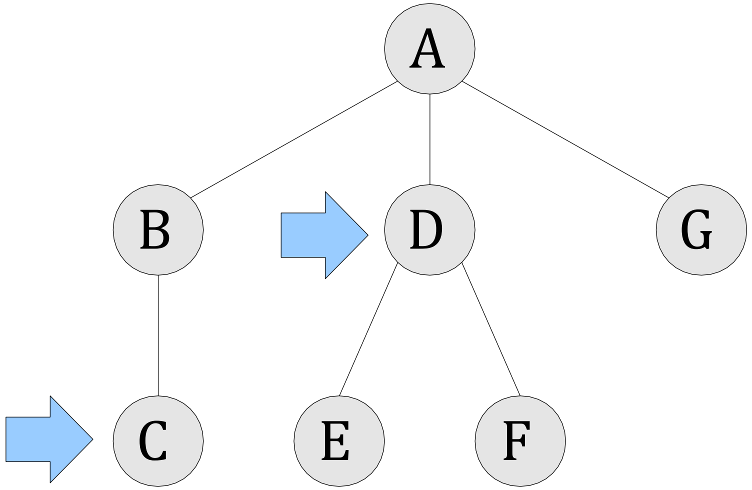
A	B	C	C	B	A	D	E	E	D	F	F	D	A	G	G	A
0	1	2	2	1	0	1	2	2	1	2	2	1	0	1	1	0

Lowest Common Ancestors



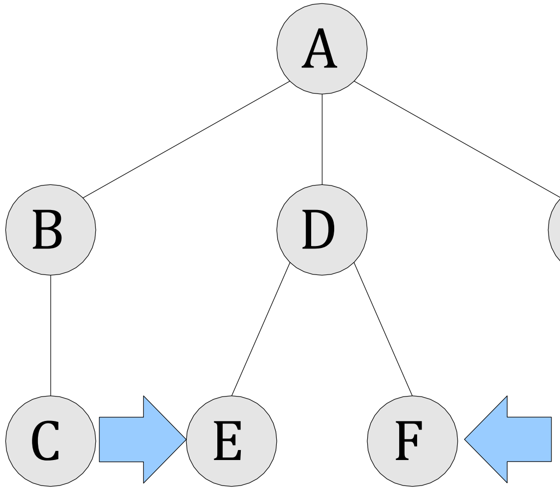
A	B	C	C	B	A	D	E	E	D	F	F	D	A	G	G	A
0	1	2	2	1	0	1	2	2	1	2	2	1	0	1	1	0

Lowest Common Ancestors



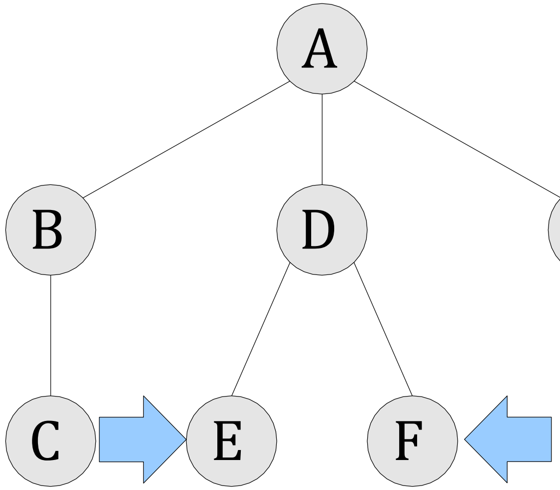
A	B	C	C	B	A	D	E	E	D	F	F	D	A	G	G	A
0	1	2	2	1	0	1	2	2	1	2	2	1	0	1	1	0

Lowest Common Ancestors



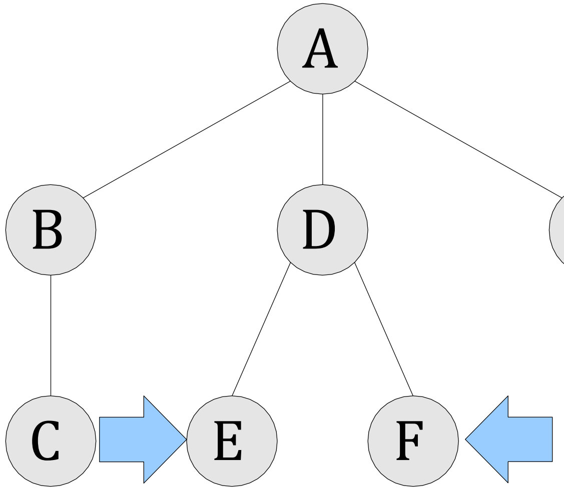
A	B	C	C	B	A	D	E	E	D	F	F	D	A	G	G	A
0	1	2	2	1	0	1	2	2	1	2	2	1	0	1	1	0

Lowest Common Ancestors



A	B	C	C	B	A	D	E	E	D	F	F	D	A	G	G	A
0	1	2	2	1	0	1	2	2	1	2	2	1	0	1	1	0

Lowest Common Ancestors



A	B	C	C	B	A	D	E	E	D	F	F	D	A	G	G	A
0	1	2	2	1	0	1	2	2	1	2	2	1	0	1	1	0

Summary

- Sparse table is a data structure, which can efficiently answer RMQ query

- LCA problem can be converted to a RMQ problem on the Euler tour sequence

- By using sparse table, we can get O(nlogn) preprocessing and O(1) querying time for both RMQ and LCA problems